GHOSTLAND

IN3026 ADVANCED GAMES TECHNOLOGY COURSEWORK

Petru Botnar
Petru.botnar.1@city.ac.uk

Contents

Project Overview

The title of the game is Ghostland, as the name suggest it involves a world populated by ghosts. In this world set in a distant space dimension, the main character must survive against the attacking ghosts. The enemies spawn in waves and are teleported through a portal coming from an unknown location. Ghostland hosts a horror themed survival third person shooter using 3D Pacman-like characters. The main objective is surviving as long as possible, in fact the game can be seen as an alternative zombie survival game. The difficulty increases as the wave count rises, apart from the number of enemies spawned, the fog level is also affected as well as the movement speed of the enemies. The gameplay offers a variety of pickups that involve health bonuses, force field grenades that would slow down enemies and gun upgrade combos that can upgrade the gun up to 5 shooting barrels! Featured with different techniques of special effects and different sound effects, Ghostland offers an exciting entertainment filled with fun, suspense and scary moments.

Assets and Libraries

All the assets and libraries have been used under freeware, open-source, GNU, GPL, BSD licenses.

Meshes

Ghosts: https://www.models-resource.com/ds_dsi/pacmanworld3/model/13183/

Lighthouse: http://www.turbosquid.com/FullPreview/Index.cfm/ID/490836

Grenade: http://www.turbosquid.com/3d-models/normal-max-free/381801

Main character: a courtesy from a friend

Textures:

Skybox: (ame_nebula) http://www.custommapmakers.org/skyboxes.php

Terrain: (hull_h)
http://www.crystalspace3d.org/svn/cc/trunk/artsource/levels/terrain%20with%20station/textures/

Pickups:

http://opengameart.org/node/8918

http://www.art.eonworks.com/free/textures/funky_texture_base_036.png

http://opengameart.org/node/7152

Animations:

Portal: https://vxresource.wordpress.com/heal_01/

Explosion: (from Template)

Audio:

https://www.freesound.org/people/Quonux/sounds/166418/

https://www.freesound.org/people/ludist/sounds/204463/

https://www.freesound.org/people/harri/sounds/9203/

https://www.freesound.org/people/Vanturus/sounds/259100/

https://www.freesound.org/people/juancamiloorjuela/sounds/204318/

https://www.freesound.org/people/B_Lamerichs/sounds/264053/

https://www.freesound.org/people/epicdude959/sounds/352508/

https://www.freesound.org/people/yummie/sounds/209731/

https://www.freesound.org/people/xDimebagx/sounds/206176/

https://www.freesound.org/people/Cyberios/sounds/145788/

https://www.freesound.org/people/josepharaoh99/sounds/364929/

https://www.freesound.org/people/jivatma07/sounds/76234/

https://www.freesound.org/people/SamsterBirdies/sounds/368492/

https://www.freesound.org/people/josepharaoh99/sounds/368591/

Libraries:

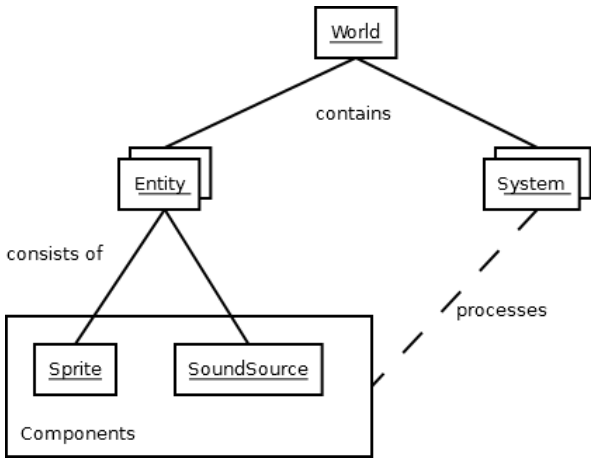
Apart from the libraries given with the template, no additional libraries were used.

Feature implementation

Entity-Component architecture

The underlying game architecture used to develop the game was implemented following the Entity-Component pattern that is Data Driven Oriented (DDO) opposed the Object Oriented (OO) used in the given Template code. Studying online material (Gaul, 2013) (Romeo, 2015) (Delgado, 2016) I have implemented a simple Entity-Component System.

The design is based on the fact that data is stored in components and entities (game objects) are collections of different components. The logic is contained within systems, which act upon the composition of the entities and update them accordingly.



This becomes a very flexible approach, where new objects and behaviour can be created in a fast and easy way.

Part 1: Basic game modelling

Intro screen and controls listing

Multiple screens have been implemented in the game using a Finite State Machine (GameFSM.h), the FSM contains the states the game can be in at any point in time, these are:

- Main Menu
- Play
- Pause
- Game Over

The UIRenderSystem would render a different screen depending on the state the game is in.

Main menu screen is our intro screen, it contains a background image that is rendered using an orthographic projection with a 2D texture mapped over a quad of the screen size. Having the GL_DEPTH_TEST disabled with can then render text on top of it.

Controls are rendered using bitmap string rendering, with given position coordinates and rgb color values.

The UI elements of other screens are rendered using the same technique.

Primitive based objects

The game contains 4 different objects created from primitives. For each object face normals for light shading are calculated and applied as well as texture coordinates on all vertices. Objects are rendered using GL_CULL_FACE enabled to avoid rendering back faces. Objects are position, rotated and scaled according to their transform component.

1. Octahedron used for the gun upgrade pickups, it is constructed with 8 triangles using GL_TRIANGLES.
2. Pentagon based pyramid used for grenade pickups, constructed using GL_POLYGONS for the base and GL_TRIANGLES for each face. (GL_TRIANGLES_STRIP could have been used here as an alternative);
3. Prism used for health pickups, constructed using GL_TRIANGLES for the sides and GL_QUADS for the other 3 faces;
4. Wall used for wall, faces constructed using GL_QUADS;

Audio

When the AudioSystem is initialised, all the audio files are loaded and stored in a map with their corresponding enum SoundID value (an error message is prompt if the loading fails), optionally it is possible

to load the file as a 3D sound by setting the argument *is3D* to true. The system listens to SoundEvent(s) that contain the *SoundID* to be played and, optional, the position for 3D audio effects. When an event is queued in the world, the AudioSystem would play it using a 3D technique if a position has been given otherwise as a normal sound. FMOD system is used to handle sounds.

3D sound loading example:

```
LoadAudioFile(SoundID::GrenadeExplosion,"Resources\\Audio\\grenade_explosion.mp3", true);
```

Sound events are synchronized with events such as:

- Entering game screens Main menu, Play and Game over;
- In game sounds attached to events such as shooting, pickups, player hit, enemy death, enemy spawns, wave begin;

Heads up display (HUD)

HUD is displayed using TextComponents, these contain all the data needed to display text on the screen

```
//Defines a text rendered on the screen
class TextComponent : public IComponent
{
public:
	explicit TextComponent(string text, float x, float y, float r, float g, float b)
	{
		this->text = text;
		this->x = x;
		this->y = y;
		this->r = r;
		this->g = g;
		this->b = b;
	}

	~TextComponent(){};

	string text;
	float x, y, r, g, b;
};
```

The UIRenderSystem, as it is responsible to render all kind of UI elements, it renders all text components data on alive entities. The values to be rendered are updated in game synchronized with different events. The GameReactionSystem creates an entity with a text component during initialisation for each of the HUD elements, and updates it on different occasions. HUD elements displayed:

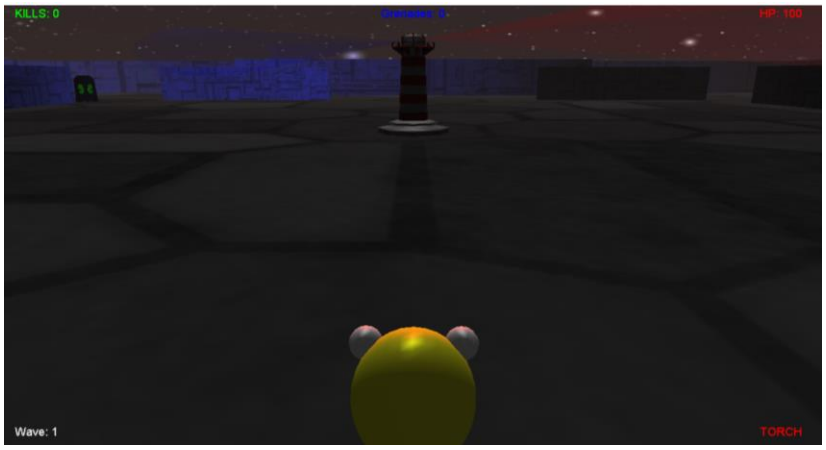
- HP;
- Kill Score;
- Grenades count;
- Torch (toggle);
- Wave number;
- Grenade throwing force;

Note: Instead of creating an entity for each HUD element (as I did) I could have simply display the value directly from the Player entity in the UIRenderSystem.

Part 2: Camera, meshes, lighting and FX

Camera motion

The CameraSystem handles camera motions, upon initialisation the system would store a pointer to the player's transform component which holds its position and direction. Camera is positioned behind the player and rotated to face its back giving a third person feeling. The position and direction of the camera is updated as the players transform data changes. Here is the result:



Meshes

Meshes are loaded and stored in a similar way to the audio files i.e. mapped using an enum id, meshes are mapped to a *MeshID.* WorldRenderSystem handles game object rendering, if the an entity contains a Prefab or Mesh component it will render it according to its enum ID value.

Textures are loaded during initialisation and the corresponding *UINT* value is stored for later usage when binding textures.

Materials are applied for each different object with custom shading properties.

Lighting

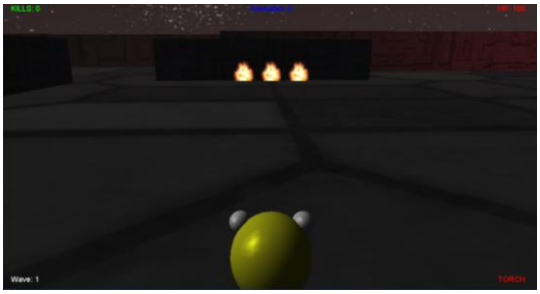
The LightingSystem controls all the lights in the game, currently there are 4 lights.

- Default light: a low strength directional light that emits grey light, that gives a nice atmosphere for a horror themed game;
- Lighthouse: contains 2 spotlights that point in opposite directions, one is red and the other is blue. The direction of the lights is updated per each frame to rotate around Y axis and give a lighthouse effect.
- Torch light: white coloured spotlight that follows the player's position and direction, it is toggled upon a TorchToggleEvent;

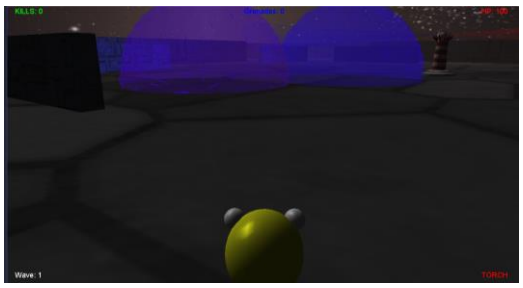
Special effects

The FXRenderSystem is responsible for rendering special effects used in game. Different types of effects are triggered using events, special effects that can be triggered are:

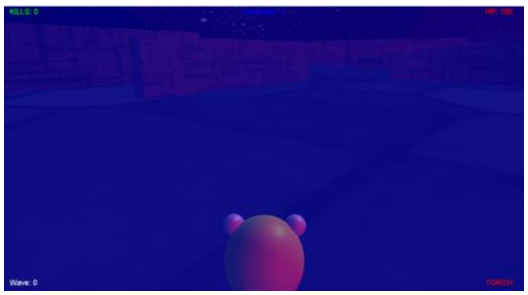
1. Sprite animations: Loaded and mapped at initialisation using an *AnimationID* and activated when AnimationFXEvent(s) are received, these hold the ID and position. To allow the same animation to be rendered multiple times at the same time in different positions, a list of active animations is maintained, upon an event a copy of the animation is added with the given properties. After rendering, any non-active animation will be removed from the active animations container.



2. Alpha spheres: Rendered when AlphaSphereFXEvent(s) are received, using the same technique as with the animations; a list of active alpha spheres is updated at each frame and cleaned up at the end;



3. Crossfades: Triggered when CrossfadeFxEvent(s) are received, using 2D rendering and orthographic projection, a full screen quad with alpha blending is rendered for a 1 second of the given rgb colour values.



Part 3: Physics, AI, and gameplay

Physics

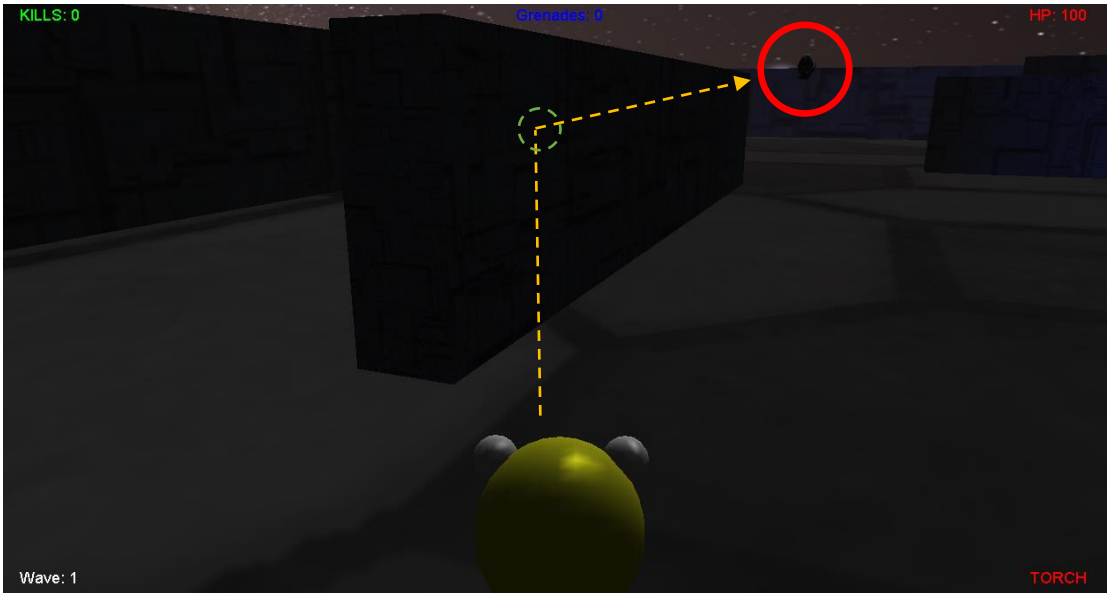
The PhysicsSystem manages all the physics used in the game, it updates the position of objects that contain a RigibBodyComponent. RigidBodyComponent(s) contain all the physical properties of the object. In the system's update method if all rigid bodies will be updated accordingly:

```
void PhysicsSystem::UpdateRigigBody(TransformComponent* transform,
RigidBodyComponent* rigidBody, float dt)
{
	// Update physical quantities

	//apply gravity
	rigidBody->m_acceleration = CVector3f(0.0f, -GRAVITY, 0.0f);

	//update position from velocity
	transform->Position += rigidBody->m_velocity * dt;

	//update rotation angle
	transform->Rotation += rigidBody->m_angularVelocity * dt;

	//update velocity from acceleration
	rigidBody->m_velocity += (rigidBody->m_acceleration + rigidBody-
>m_instantaneousAcceleration) * dt;

	//update rotation velocity
	rigidBody->m_angularVelocity += (rigidBody->m_angularAcceleration + rigidBody-
>m_instantaneousAngularAcceleration) * dt;

	// Turn off instantaneous forces if contact time is surpassed
	if (rigidBody->m_instantaneousAcceleration.Length() > 0 && rigidBody-
>m_contactTime > 0.05) {
		rigidBody->m_instantaneousAcceleration = CVector3f(0, 0, 0);
		rigidBody->m_instantaneousAngularAcceleration = CVector3f(0, 0, 0);
		rigidBody->m_contactTime = 0;
	}
	rigidBody->m_contactTime += dt;

	//chech and respond with planes collision (terrain/walls)
	if (IsCollisionOnPlane(rigidBody->m_radius, transform->Position, rigidBody-
>m_velocity))
	{
		RespondToPlaneCollision(CVector3f(0, 1, 0), transform->Position,
rigidBody);
	}
}
```

CollisionsDetectionSystem is implemented using the AABB collision detection approach, the bounding box properties are stored in BoxColliderComponent(s). The system iterates through all entities that contain a BoxCollider in a nested loop and checks for a collision. To avoid checking the same collision twice, the second loop is iterated backwards starting at an offset location where the offset is equal to the number of iteration of the top loop.

When a collision is detected, a CollisionEvent is sent in the world with the two colliding entites, the normal face and depth of the collision.

Applied physics when throwing grenades:



Grenade bouncing off the wall

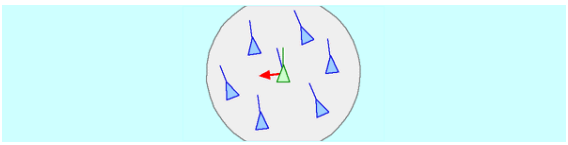
Artificial Intelligence

The AISystem controls the behaviour of all NPC characters (ghosts) present in the game. AI is implemented using steering behaviours such as flocking, seeking and object avoidance. Following the guides and tutorial on steering behaviours (Reynolds, 1999) (Pemmaraju, 2013) (Bevilacqua, 2013).

- **Flocking**

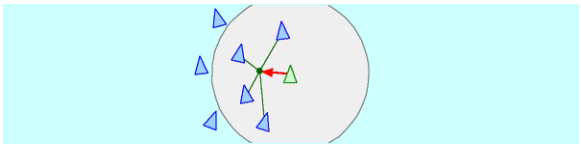
  flocking is contructed using 3 elements:

  1. Alignment: a behaviour that makes an agent line up with its neighbours within a range, it is implemented by averaging the velocity of all the neighbours;



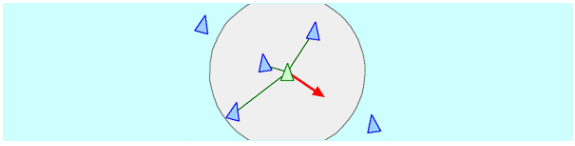
Adapted caption from Reynolds

  2. Cohesion: a behaviour that makes agents steer towards the centre of the neighbourhood, it is implemented by averaging all the positions of the neighbours;



Adapted caption from Reynolds

3. Separation: a behaviour that makes the agent steer away from its neighbours; calculated by averaging the differences of in position over the distance of the neighbours;



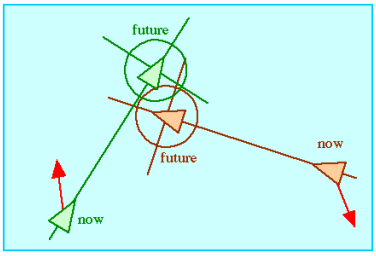
Adapted caption from Reynolds

- **Seeking:**

  Calculated by calculated the direction towards the target;

- **Object avoidance:**

  Calculated by checking if a collision occurs using future positions, and steering using a force that point away from the colliding object;



Adapted caption from Reynolds

For each steering behaviour, there is a constant weight value applied; by tweaking this values it is possible to have different behaviour results;

Gameplay features

The gameplay features many elements such as different pickup types, increasing level of difficulty and timed events.

- **Pickups** (spawning with a probability chance after killing an enemy)
    - Health bonus pickups: increase player's HP value by +25;
    - Gun upgrade: a combo that when picked adds 2 gun barrels up to 5, when hit by an enemy the combo is lost and the player returns to 1 as default;
    - Force field grenades: grenades that explode after 3 seconds and creation a force filed that slows down any enemy within its range;
- **Level difficulty**
    - Enemies spawn in greater quantities and they get faster;
    - Rising fog level;
- **Horror elements**
    - Enemies within a certain range may start screaming (timed events);

Discussion

Given such a short amount of time, the result is acceptable. More gameplay elements could be introduced to offer a more challenging experience such as:

- Ammo: the player has an ammo score and needs to reload and collect ammo pickups;
- Extra lives: permitting the player to continue playing after dying;
- Multiplayer feature (advanced): implement a network system that sends and receives game events such as as position updates, etc. Using the current design this can be achieved without massive changes to the code;

In terms of game design the collision detection approach becomes very expensive whit large numbers of game objects, an algorithm such as SweepAndSort or space partitioning could be used to detected whether a collision may be occurring in a certain area.

References

Bevilacqua, F., 2013. *gamedevelopment.tutsplus.com.* [Online]
Available at: https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-collision-avoidance--gamedev-7777

Delgado, J., 2016. [Online]
Available at: https://github.com/JuDelCo/Entitas-Cpp

Gaul, R., 2013. *http://www.randygaul.net/.* [Online]
Available at: http://www.randygaul.net/2013/05/20/component-based-engine-design/

Pemmaraju, V., 2013. *gamedevelopment.tutsplus.com.* [Online]
Available at: https://gamedevelopment.tutsplus.com/tutorials/3-simple-rules-of-flocking-behaviors-alignment-cohesion-and-separation--gamedev-3444

Reynolds, C. W., 1999. *http://www.red3d.com/.* [Online]
Available at: http://www.red3d.com/cwr/steer/gdc99/

Romeo, V., 2015. [Online]
Available at: https://www.youtube.com/watch?v=NTWSeQtHZ9M

ECS image: http://python-utilities.readthedocs.io/en/latest/_images/ebs.png